# Towards Self-tuning of Dynamic Resources for Workloads

Fu Duan, Yongjie Han, Qiuyong Zhao, Keming Xie
*Taiyuan University of Technology, Taiyuan, Shanxi, P.R.China, 030024*
*duanfu@tyut.edu.cn, hhyyjj0000@126.com*

## Abstract

*In self-tuning of database performance research area, the optimization of dynamic resources is important. This paper focused on the self-tuning of dynamic resources on the base of feedback mechanism, designed and achieved a simple model combined with Oracle database. This model gave an evaluating principle of system performance, and could adjust system parameters automatically to achieve the goal of system performance. And an evaluation model was designed to evaluate the performance state of the running system. The result after the self-tuning of some parameters was illustrated by simulating various workloads.*

## 1. Introduction

Self-tuning technology for database management systems (DBMS) is becoming increasingly important to database performance. Because the cost of maintaining database is increasing and the manual tuning is hard to adapt to the changing of complex workloads, self-tuning technology can automatically adjust the performance of database according to the need of application and the ability of hardware. Dynamic Resources tuning is one aspect of self-tuning technology. The purpose is adjusting automatically the various parameters of resources to achieve the goal of system performance on the running of system.

Resources of database system can be divided into physical resources and logical resources. Physical resources are the hardware aspects, and their values are limited by the condition of hardware, such as CPU, memory, disk, etc; logical resources are the DBMS provides, and their values are limited by the DBMS, such as the number of work threads, the number of locks, etc [4]. Physical resources and logical resources are interactive and interdependent. It requires a detailed understanding of the nature of activities that compete for the resources being managed, including storage space and the I/O channel. Complex query processing will result in widely varying memory, processor, and disk demands. Self-tuning the performance of the DBMS is becoming a complex task because of the complex interactions and different effects on system performance among these resources [3]. Generally speaking, the internal algorithms are driven by some adjustable parameters provided by DBMS. The values of these parameters have effect on the performance of the target system. On the running of system, the self-tuning of parameters with the changing of various workloads could have effect on the performance of system in some extent.

In this paper, we built a prototype on Oracle database system based on the feedback mechanism. The main performance parameters of the target system are dynamically adjusted according to the dynamic performance views and data dictionaries. And an evaluation model was designed to evaluate the performance state of the running system. The result after the self-tuning of some parameters was given by simulating various workloads running on the target system.

This rest of this paper is organized as follows. Section two gives a brief review of some previous work related to database performance tuning; Section three introduces the framework of the feedback mechanism and presents the architecture of optimization prototype; Section four presents the evaluation principle and the objective formula of performance. Section five gives the plan of diagnosis and adjustment; Section six illustrates the result of self-tuning of database system; and Section seven gives conclusions and future work.

## 2. Related Work

In the area of self-tuning of database performance, there are more researches on automatic memory allocation. The way to self-tuning of the memory contains automatic allocating buffer pool, using memory in distributed system, pre-fetching data, etc. The mainly focused algorithm is the scheduling of buffer pool, such as LRU-K page swapping algorithm,


E-FORENSICS 2008, January 21-23, Adelaide, Australia

DOI 10.4108/wkdd.2008.2653

Class fencing, Dynamic Reconfiguration(DRF), the configuration of multiple buffer pools algorithm, etc [2]. The goal of these algorithms is improving the buffer hit rate.

An algorithm proposed in [3] that automatically adjust both multiprogramming levels and memory allocation to achieve a set of per-class response time goals for complex workloads in DBMS. It uses a set of heuristics and estimation techniques to control a feedback mechanism. A Buffer Pool Tuning Wizard is described in [6] to determine effective buffer pool sizes. The wizard is based on a self-tuning algorithm called the Dynamic Reconfiguration algorithm (DRF), which uses greedy heuristics to find a reallocation that benefits a target transaction class. The framework for workload adaptation is proposed in [1]. It uses performance objective encapsulation techniques to combine individual performance objectives into an objective formula of performance. It is optimized based on a performance model to find a solution for workload control. Menascé [5] describes an approach in which analytic performance models are combined with combinatorial search techniques to design controllers that run periodically to determine the best possible configuration for the system given its workload. A new approach of self-managing computer system is built into the system, the mechanisms required to self-adjust configuration parameters so that the Quality of Service requirements of the system are constantly met.

## 3. Framework for Optimization

Feedback mechanism could be as the fundamental architecture of the self-tuning system. In this heuristic mechanism, the main components are as follows: performance monitor, performance analyzer and performance optimizer. System begins to run with the predefined configure parameters. The monitor module monitors the various performance information of the target system constantly during the running period, and passes the results to the analyzer module. If the analyzer module finds the performance of the target system down to the predefined thresholds, it will send some instructions to some parameters need to be adjusted. After adjusting these parameters, if the goal of the performance could not be met, system will enter the recycle progress of observing, analyzing and adjusting [7].

The architecture of the optimization prototype is best described in figure 1. The modules of the framework are performance monitor, performance evaluation, resources diagnosis, the knowledge set of rules, performance analyzer and resources adjuster.

Performance monitor dynamically monitors the performance of the target system with the changing workloads. It executes at regular intervals with the execution of controller algorithm and collects the resource utilization and the performance state by the dynamic views and data dictionaries on the system running. Performance evaluation evaluates the performance of the current system. And it gives out the information whether to optimize the system based on the principle of the evaluation model. Resources diagnosis finds out the bottleneck of resources and decides whether the DBMS needs to be tuned. The knowledge set of rules is used to store the rule's set of performance optimizing. The knowledge set provides information for the diagnosis and analyzer modules giving out the plan of adjusting the parameters of performance resources. Performance analyzer decides how to adjust the parameters and gives out the optimization plan in the valid range of the parameters. Resources adjuster automatically adjusts the specific parameters of performance resources according to the real value passed by the analyzer module.

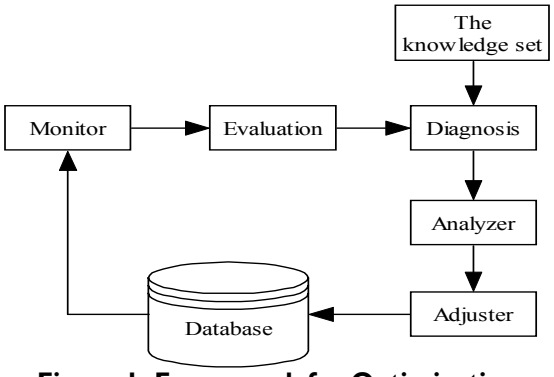


**Figure 1. Framework for Optimization**

## 4. Evaluation and Analysis

The main work of designing evaluation is how to evaluate these resource indexes. In order to simplify the work, we choose the main resources that affect system performance significantly, including transaction throughput, cache hit ratio, response time, the ratio of request refusing and the utilization ratio of CPU. We built the database performance index to quantify these resources. Database performance index is illustrated in table 1.

We give every performance index a mount of scores which presents the state of current system performance. As the different requirement of every performance index, the score is a little difference among these indexes. Taking buffer cache hit ratio as an example, the specific evaluation principle is illustrated in table 2.

**Table 1. Database Performance Index**

| Type | Index | Score |
|---|---|---|
| Response Time | >85% | 100 |
| Request Refusing | >85% | 100 |
| Transaction Throughput | >85% | 100 |
| Utilization ratio of CPU | >85% | 100 |
| Buffer Cache Hit Ratio | >95% | 100 |
| Library Cache Hit Ratio | >98% | 100 |
| Dictionary Cache Hit Ratio | >98% | 100 |
| Sort in Memory | >98% | 100 |
| Log Buffer Hit Ratio | >98% | 100 |

**Table 2. Evaluation Principle**

| Index | Score |
|---|---|
| >98% | 100 |
| 95-98% | 80 |
| 90-94% | 60 |
| <90% | 0 |

Generally speaking, the performance is better when the score is higher. The score is zero illustrates some part of system has problem. According to the performance index, we could get the current state of system from statistic information and use this evaluation model to give out the state of system performance. The objective formula is defined as,

$QOS = r_1*(1 - R_0 / R_{max}) + r_2*(1 - P_0 / P_{max}) + r_3*(1- X_{min} / X_0) + r_4*(1 - CPU_{min} / CPU_0) + r_5*(a_1*Rate_1+ a_2*Rate_2+...+ a_n* Rate_n)$

where $r_i$ ($r_i \geqslant 0$, and $r_1 + r_2 + r_3 + r_4 + r_5 = 1$) presents the weight indicating the relative importance of each performance index. We define $R_0$ as the average response time, $R_{max}$ as the longest one tolerated, $X_0$ as the average throughput of system, $X_{min}$ as the minimum one tolerated, $P_0$ as the average refusing ratio of request, $P_{max}$ as the maximum one tolerated, $CPU_0$ as the average ratio of CPU utilization, $CPU_{min}$ as the minimum one tolerated, $Rate_i$ as the various cache hit ratios, $a_i$ as the weight ($a_i \geqslant 0$ and $a_1 + a_2 + \ldots + a_n = 1$). The QOS metric means the quality of service of target system. $R_0$, $P_0$, $X_0$, $CPU_0$, $Rate_i$ could be used as the main indexes to measure the utilization ratio of system resources.

As the workloads change constantly, the performance of system vary accordingly. The problem occurred should be solved timely, so the algorithm can not be complicated. We use the algorithm called hill-climbing to optimize the parameters of system performance. The algorithm is simply described here. H (h1, h2, h3) is a vector set where h1, h2, h3 are the adjustable parameters may have effect on the system performance. Every vector has the maximum and minimum values. A "neighbor" value of H is defined as one of vector $h_i$ is changed by +1 or –1. We define V = (h, h, h) as a variable vector set. In the vector set V, there is only one h is 1, the other is 0. Taking V = (1, 0, 0) for example, H+V means the value is (h1+1, h2, h3). The search is repeated at each new vector visited until the requirements of the QOS and each performance index are met.

## 5. Experiments

We developed self-tuning program, and tried to make it running with a little system resources consumption. In order to testify the effectiveness of the self-tuning system, we developed series of experiments to compare the performance with self-tuning and without tuning of database system. We use TPC-C benchmark as our workloads. It uses some TPC-C transactions, in order to generate a mixed-transaction called OLTP (Online Transaction Processing). The workload with multi-sessions is produced by a series of queries submitted by clients periodically.

As the figure illustrates below, response time presents the time of certain series of transactions executed, sessions present different numbers of clients connected to the server, TXN presents different numbers of transactions executed. When the number of sessions or the transactions is few, the response time of execution is no difference with and without the self-tuning of system. However, the response time increases rapidly without the self-tuning of system. But the time is controlled well under the self-tuning of system.

As figure four and five illustrate, TPS presents transactions per second. It gives the reference of the system throughput. Different sessions executing with certain amount of transactions have different TPS. The value of TPS is great as the number of sessions increasing until up to the limit ability of the system. With the self-tuning of database system, the TPS is higher than the one without the self-tuning of system as the number of sessions increasing. When the number of transactions is few, the TPS is almost the same as the one without the self-tuning of system. However, when the number of transactions increasing, TPS with the self-tuning of system increases faster than the one without the self-tuning of system. And TPS without the self-tuning of system keep in certain level of small value.

After the self-tuning of system, the response time reduces and the TPS increases in every group. It signifies that the performance of system has some improvements by the self-tuning of performance resources.

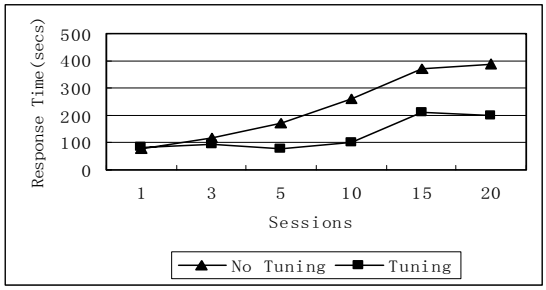


**Figure 2. Response time for different sessions**

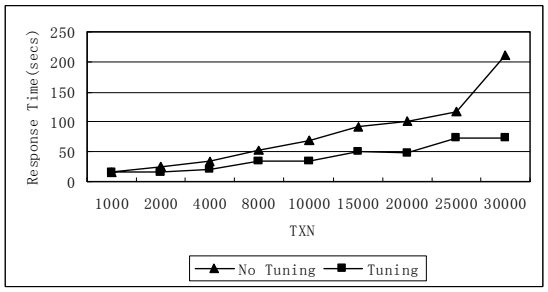


**Figure 3. Response time for different transactions**

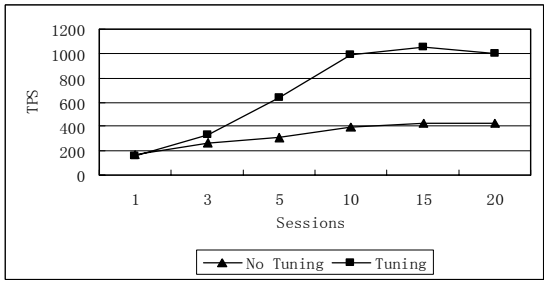


**Figure 4. TPS for different sessions**

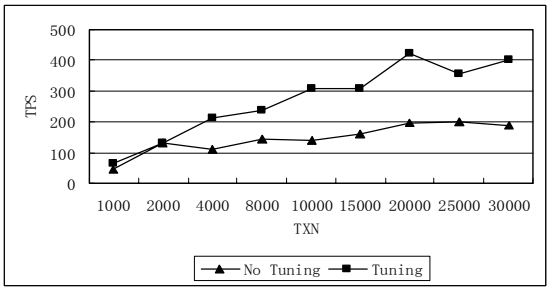


**Figure 5. TPS for different transactions**

## 6. Summary and Future Work

In this paper we presented a framework and prototype implementation for the self-tuning of database performance resources. We used the feedback mechanism to monitor, diagnose, evaluate, analyze and adjust the target system on the running.

For future work, we plan to improve the policy of adjustment, introducing some optimal algorithms to make better optimization plan. We expect to detail the goal of performance optimization. Self-tuning resources allocated to every session connected to database in order to satisfy the requirements of different users.

## 7. Acknowledgments

The work is supported by the Returned Student Research Foundation of Shanxi Province both (2004-18) and (2004-26), the National Special Science Foundation (2006FY11070903), and the Natural Science Foundation of Shanxi Province (No. 200601103).

## 8. References

[1] Baoning Niu, P. Martin, W. Powley, R. Horman and P. Bird, "Workload Adaptation in Autonomic DBMSs", Proceedings of CASCON, 2006, pp. 161 - 173.

[2] D.G. Sullivan, M.I. Seltzer, A. Pfeffer, "Using probabilistic reasoning to automate software tuning", International Conference on Measurement and Modeling of Computer Systems, 2004, pp. 404-405.

[3] K. P. Brown, M. Mehta, M. J. Carey, and M.Livny, "Towards Automated Performance Tuning For Complex Workloads", Proceedings of the 20th Very Large Data Base Conference, Santiago, Chile, 1994.

[4] Ivan T. Bowman, David Toman Abdelkader Hameurlain, F. Morvan, "CPU and incremental memory allocation in dynamic parallelization of SQL queries", Parallel Computing, 2002, pp. 525–556.

[5] D. A. Menascé, and M. N. Bennani, "On the Use of Performance Models to Design Self-Managing Computer Systems", Proceedings of 2003 Computer Measurement Group Conference, Dallas, TX. USA, Dec.7-12, 2003, pp. 1-9.

[6] P. Martin, W. Powley,Zheng Min, K.Romanufa, "Experimental study of a self-tuning algorithm for DBMS buffer pools", Journal of Database Management, Apr.2005, pp. 1-20.

[7] D. Botzer, and O. Etzion, "Self-Tuning of the Relationships among Rules' Components in Active Databases Systems" IEEE Transactions On Knowledge And Engineering, Mar.2004, pp. 375-379.